Topics for today

- games in AI: What can be achieved and why one should care
- state-spaces in games
- adversarial search for game-play
- outlook on selected advanced game-playing methods

Educational objectives: being able to...

- evaluate describe of AI research in games
- describe games as state-space search
- characterise games by applicability of methods
- implement state-space search methods $\text{MinMax}$, $\alpha-\beta$-pruning, and MCTS
**Games** have always been considered as a topic for AI research

- good game-play in games like chess or Go is very hard for humans, yet the problem can be easily represented
- performance of AI programs easily comparable to humans
- characterised by Jonathan Schaefer as “microcosm for AI research”
- relevant real-world applications can be modelled as games
- they are much harder to solve than search problems considered so far
  ∼ We aren’t able to explore all relevant parts of the state space!
Examples

early chess computer
photo: Thorsten Czub

AI Birds

Robocup soccer
photo: RoboCup2014.org
Still, researchers are interested in developing better-performing algorithms for special games
- convenience of evaluation, good comparability
- games tailored to research problems
- investigate a particular game, e.g., existence of winning strategy (checkers turned out to be a draw!)

Rather than developing game-specific techniques, research now focuses on universal methods
- based on rules of the game, programs have to develop strategies autonomously
- **Universal Game-Playing** using the **Game Description Language (GDL)**
- **General Video Game AI Competition**

Developing means for competitive game-play i.e., fun-to-play-against computer players

Specifying games in an abstract language (e.g., GDL) is a challenging task in the area of **Knowledge Representation** – we’ll come back to that!
**Question:** What is a common feature of games like Go, chess, checkers, Tic-Tac-Toe?

**Question:** And how do these games differ from...

1. poker, Skat;
2. arcade games like Pacman or Tetris;
3. snooker, soccer, or tennis?
Games can be categorised by decisive characteristics:

- number of players involved and whether they are cooperative or adversarial
  - cooperative games are considered in the domain of multi-agent systems
- state space is fully observable or partially unknown
- state space is discrete or continuous
- observations and actions are deterministic or stochastic
- game is concurrent or round-based

Today, we will focus on two-player adversarial games with fully observable state space and deterministic observations and actions.
<table>
<thead>
<tr>
<th></th>
<th>Chess, Go, etc.</th>
<th>poker</th>
<th>arcade video games</th>
<th>Angry Birds</th>
<th>RoboCup soccer</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>environment</strong></td>
<td>static</td>
<td>static</td>
<td>dynamic</td>
<td>piecewise dynamic</td>
<td>dynamic</td>
</tr>
<tr>
<td><strong>state transition</strong></td>
<td>round-based</td>
<td>round-based</td>
<td>continuous</td>
<td>round-based, piecewise continuous</td>
<td>continuous</td>
</tr>
<tr>
<td><strong>knowledge</strong></td>
<td>total</td>
<td>partial</td>
<td>total</td>
<td>partial</td>
<td>partial</td>
</tr>
<tr>
<td><strong>observation</strong></td>
<td>symbolic, discrete</td>
<td>symbolic, discrete</td>
<td>non-symbolic</td>
<td>non-symbolic</td>
<td>non-symbolic</td>
</tr>
<tr>
<td><strong>actions</strong></td>
<td>deterministic</td>
<td>deterministic</td>
<td>deterministic</td>
<td>little uncertain</td>
<td>uncertain</td>
</tr>
<tr>
<td><strong>super-human AI?</strong></td>
<td>yes</td>
<td>yes</td>
<td>yes</td>
<td>no</td>
<td>no</td>
</tr>
</tbody>
</table>
Objective is to master a **physical simulation game**

In contrast to arcade video games and universal video game playing, Angry Birds involves many strategic elements and uncertainty

Conjectured to be next breakthrough on way to real-world AI

Where should the bird be aimed at to destroy all pigs?
• Objective is to master a **physical simulation game**
• In contrast to arcade video games and universal video game playing, Angry Birds involves many strategic elements and uncertainty
• Conjectured to be next breakthrough on way to real-world AI
Topic of today’s session: learn basics that allow you developing super-human game play in classic strategy games like chess.

- State-space search is important for mastering games with discrete state spaces
- However, BFS or A* cannot directly be applied to compute good moves: both players pursue contrary goals!

⇝ adversarial search
• We consider a round-based game of two adversarial players, called challenger and opponent.

• **Definition:** the game tree is a tree comprising all possible game states, the game’s start state is the root of the tree. The tree is constructed by alternating challenger and opponent moves.
Example Game Tree
Observations

- Game trees are too big to depict, even for a simple game (neat idea: https://xkcd.com/832/)
- State expansion from tree nodes at depth $2n$ to depth $2n + 1$, $n \in \mathbb{N}$ corresponds to move of the challenger
  - recall: **depth** measured by distance from root
- Game trees may not be trees: in some games, multiple move sequences can lead to the same game state
  - **examples**: tic-tac-toe, chess, checkers, ...
  - from an algorithmic point of view, we *may* treat such states as distinct (hence we stick with ‘tree’)
  - **hint**: a good implementation will recognise it has seen the state before (caching)!
• Consider the game tree of a two-player adversarial game

• **Objective:** from a given node of the game tree, compute a move that will lead to the best possible outcome
  • win (as high as possible), if possible
  • draw or (least possible) defeat otherwise

• **Idea:** assume opponent to play optimal, then opponent follows strategy opposite to player
Consider the game tree of a two-player adversarial game

**Objective:** from a given node of the game tree, compute a move that will lead to the best possible outcome
- win (as high as possible), if possible
- draw or (least possible) defeat otherwise

**Idea:** assume opponent to play optimal, then opponent follows strategy opposite to player

**Approach:** fully expand the game tree, label leaves in the game...

+1 if the challenger wins (by the winning score, respectively)
0 for a draw
−1 if the challenger wins (by the score lost, respectively)
Consider the game tree of a two-player adversarial game.

**Objective:** from a given node of the game tree, compute a move that will lead to the best possible outcome

- win (as high as possible), if possible
- draw or (least possible) defeat otherwise

**Idea:** assume opponent to play optimal, then opponent follows strategy opposite to player

**Approach:** fully expand the game tree, label leafs in the game...

- +1 if the challenger wins (by the winning score, respectively)
- 0 for a draw
- −1 if the challenger wins (by the score lost, respectively)

... and propagate the information up to the root node
• $S_9 = +1$, therefore $S_8$ should be $+1$ too, since we have no choice

• **Question:** What value should be assigned to $S_5$?

$S_5 = -1$ since player X would go for the win!
• $S_9 = +1$, therefore $S_8$ should be $+1$ too, since we have no choice

• **Question**: What value should be assigned to $S_5$?

• $S_5 = -1$ since player X would go for the win!

\[
\begin{array}{c}
S_0 \\
\text{turn:} \quad \bullet \\
\hline
S_1 \\
\text{turn:} \\ x \\
\hline
S_2 \\
\text{turn:} \\
+1 \\
\hline
S_3 \\
\text{turn:} \\
\hline
S_4 \\
\text{turn:} \\
S_5 \\
\text{turn:} \\
-1 \\
\hline
S_6 \\
\text{turn:} \\
\hline
S_7 \\
\text{turn:} \\
\hline
S_8 \\
\text{turn:} \\
\hline
S_9 \\
\text{turn:} \\
+1
\end{array}
\]
• $S_9 = +1$, therefore $S_8$ should be $+1$ too, since we have no choice

• **Question:** What value should be assigned to $S_5$?
• $S_5 = -1$ since player X would go for the win!

• **Question:** What value should be assigned to $S_3$?
• $S_9 = +1$, therefore $S_8$ should be $+1$ too, since we have no choice
• **Question:** What value should be assigned to $S_5$?
• $S_5 = -1$ since player X would go for the win!
• **Question:** What value should be assigned to $S_3$?
• $S_3 = +1$ since player O would go for the win!
• $S_9 = +1$, therefore $S_8$ should be $+1$ too, since we have no choice
• **Question:** What value should be assigned to $S_5$?
• $S_5 = -1$ since player X would go for the win!
• **Question:** What value should be assigned to $S_3$?
• $S_3 = +1$ since player O would go for the win!

⇝ Challenger **maximises** the result, opponent **minimises** the result
function MINIMAX(N, p) ▽ determines best move for player p

1. Children ← EXPANDSTATE(N, p)
2. if Children = ∅ then ▽ leaf?
3. return EVALUATESTATE(N), N)
4. else
5. Moves ← {(v, c)|c ∈ Children, v = MINIMAX(c, −p)}
6. if p = +1 then
7. return arg max(v,s)∈Moves v
8. else
9. return arg min(v,s)∈Moves v
10. end if
11. end if
12. end function
• **MinMax** evaluates a node $N$ in the game tree, assuming it is the turn of player $p$ (challenger $+1$, opponent $-1$)
  • value $+1$ represents win of player $+1$
  • value $-1$ represents win of player $-1$
• The version given here returns value $v$ and successor node $s$, so it is clear which move should be taken
  • in an implementation one may choose to return (just) the best move instead, or the overall game-play as sequence of moves

**Theorem**

**MinMax** returns $+1$ if and only if there is a winning move for the challenger, i.e., it is not possible for the opponent to win or enforce a draw.

**Theorem**

Let $(T_N, E_N)$ be game tree rooted at node $N$. **MinMax** requires $O(|T_N|)$ time.
Proof sketch.
⇒: recursively follow the successor nodes at depth $d$ of player $p \cdot (-1)^d$ to unfold the winning strategy
⇐: proof by contradiction, assume MinMax to return $+1$ although no winning move exists. This contradicts to part “⇒” of the proof.

Proof sketch.
MinMax evaluates each nodes and then considers it once when maximising (respectively minimising) its parent.

⇒ Although MinMax is linear wrt. the number of nodes, these typically grow exponential with respect to the depth of the game tree!
With MinMax we have an algorithm with optimal performance wrt. quality of game-play

- unfortunately, it is **useless** in any real application

However, all advanced approaches can be posed as improvements of MinMax

1. Better state space **representation**
2. **Pruning** off irrelevant parts of the search tree
3. **Reducing search depth** by estimating values of inner nodes instead of propagation from leafs
4. **Reducing search breadth** by focusing on promising moves

Techniques 1–2 retain optimal performance of MinMax, 3–4 are only approximations. State-of-the-art implementation often make use of all techniques.
State Space Representation

Objective is to **avoid unnecessary computations** of MinMax.

- **confluence** of paths, i.e., multiple paths leading to the same state
  - E.g., use caching to store all states explored and check after state expansion whether a state’s result is already stored

- **symmetry** of states, multiple states are conceptually similar. In Tic-Tac-Toe, boards may be rotated or mirrored
  - **transposition tables** are used to identify mirrored, rotated, ... version of the state and to translate moves from e.g. a mirrored state

- **decomposition** of game into different phases to reduce complexity
  - In chess programs, opening books are used and special methods implemented for different kinds of endgames (only kings and pawns, etc.)

All approaches except decomposition retain MinMax optimality. They are very helpful, but will not suffice for complex games.
Idea: prune off parts of the search tree known to be irrelevant

- e.g., if a better move is already known
- $\alpha - \beta$-pruning is most effective if good moves are explored early
  - after state expansion, sort states by expected utility of moves (heuristic)
- method most effective in games with fine-grained node evaluation, i.e., not just $+1, 0, -1$. 
Let’s revisit our example from \texttt{MinMax}:

- when expanding $S_0$, assume $S_2$ is checked first
- \textbf{Question}: Why doesn’t $S_1$ need to be explored further?
Let’s revisit our example from MinMax:

- when expanding $S_0$, assume $S_2$ is checked first
- **Question:** Why doesn’t $S_1$ need to explored further?
- Because $S_1$ won’t lead to a better outcome

The idea of $\alpha - \beta$ pruning is to store the best move of the challenger (opponent) in variable $\alpha$ ($\beta$) and disregard branches that will not change $\alpha$ or $\beta$. 
1: function AlphaBeta($N$, $p$, $\alpha$, $\beta$) \hfill $\triangleright$ best move for player $p$
2: \hspace{1em} Children $\leftarrow$ ExpandState($N$, $p$)
3: \hspace{1em} if Children $= \emptyset$ then \hfill $\triangleright$ leaf?
4: \hspace{2em} return (EvaluateState($N$), $N$)
5: \hspace{1em} else
6: \hspace{2em} if $p = +1$ then \hfill $\triangleright$ maximising player
7: \hspace{3em} $v \leftarrow -\infty$
8: \hspace{3em} for $c \in$ Children while $\alpha < \beta$ do
9: \hspace{4em} $v \leftarrow \max\{v, \text{AlphaBeta}(c, -p, \alpha, \beta)\}$
10: \hspace{3em} $\alpha \leftarrow \max\{\alpha, v\}$
11: \hspace{3em} end for
12: \hspace{2em} else \hfill $\triangleright$ minimising player
13: \hspace{3em} $v \leftarrow \infty$
14: \hspace{3em} for $c \in$ Children while $\alpha < \beta$ do
15: \hspace{4em} $v \leftarrow \min\{v, \text{AlphaBeta}(c, -p, \alpha, \beta)\}$
16: \hspace{4em} $\beta \leftarrow \min\{\beta, v\}$
17: \hspace{3em} end for
18: \hspace{2em} end if
19: \hspace{1em} end if
20: \hspace{1em} return $v$ \hfill $\triangleright$ Here we only return evaluation, not the best move
21: end function

Initially, AlphaBeta will be called with arguments $\alpha = -\infty$, $\beta = +\infty$
Theorem

The $\alpha - \beta$ algorithm will return the same value as MinMax, but possibly not evaluate as many nodes.

Proof idea.

We have to show that branches pruned off will not affect computation of MinMax. Observe that $\alpha$ represents the minimal score the maximising player can achieve (it’s maximised only when she gets to choose!), $\beta$ represents the maximal score the minimising player can achieve. Assume $\alpha \geq \beta$ gets satisified in the maximising strand (lines 7–10). Then we found a move to some state $c$ which yields a higher value than the minimising player could achieve in some other branch. Thus, she would choose that other branch, exploring the present branch may be aborted.
Reducing Search Depth

**Idea:** Don’t expand tree to leaf nodes, but evaluate inner nodes.

- Typically, expand tree for $d$ levels and evaluate nodes at level $d$. $\Rightarrow$ choosing $d$ controls compute time
- Unlike evaluation at leafs, evaluating inner nodes in very difficult
  - usually, heuristics are applied to assess value of a node
  - e.g., in Tic-Tac-Toe one may count how many lines are still unblocked by the opponent; in chess a dedicated value scheme has been developed (by humans for evaluating human game-play)
  - state-of-the-art systems like AlphaGo use machine learning techniques to learn an appropriate value function

$\Rightarrow$ heuristic or learnt evaluation functions can lead to imperfect game-play if evaluation is poor
Example of evaluation function considering unblocked lines, values above zero indicate a higher likelihood of maximising player to win then of the minimising player.

\( S_1 \) three unblocked lines for red (maximising) player, four for black (minimising player)

\[ v(S_1) = 3 - 4 = -1 \]

\( S_2 \) four unblocked lines for red, two for black

\[ v(S_2) = 4 - 2 = 2 \]

- However, \( S_2 \) is a clear win for the player next to move \( \sim \) example of a potential weakness of this evaluation function.
Obtaining Evaluation Functions

- manually designed evaluation functions have been successfully used in chess programs like IBM’s Deep Blue which defeated Kasparov in 1997
- weaknesses in an evaluation function lead to systematic failures of the game program
- In Google’s winning AlphaGo, a function $\text{State} \rightarrow \text{Value}$ has been fitted automatically to prepared training data
  - using 30 million hand-annotated states!
- Alternatively, one can try to learn from values propagated back once a game is decided
  - e.g., by reinforcement learning
  - which is used in AlphaGo Zero and can be trained from self-play, i.e., programs playing against themselves.
  - however, learning from self-play is not easy (getting stuck in local minima)
**Idea:** Rather than relying on a heuristic, randomly select which parts of the tree to explore and use statistics to compute expected value of moves.

- randomised algorithm, so-called **Monte-Carlo** method as it relies on chance
- implements **random sampling** of the search tree
- randomised search started to become popular in the 1990ies (IBM DeepBlue only used $\alpha - \beta$ pruning), since first decade of 2000 all state-of-the-art programs employ randomised search
- **Approach:** fix a computational budget and randomly explore the search tree. After time has run out, determine best move.
Monte-Carlo Tree Search (MCTS)

MCTS starts with a search tree only containing the current game state. That tree is expanded by repeatedly performing:

- **Selection**: select a node in the game tree explored for further investigation
- **Expansion**: expand the game tree represented by adding a new node $N$
- **Rollout**: complete one playout from node $N$ to sample how the game could develop
- **Backpropagation**: use the result of the playout to update information in the nodes on the path from $C$ to the root in order to guide subsequent selection steps

For each of the steps, different techniques can be employed – we’ll only look at the most popular choices.
Sequence is repeated until running out of time.
The objective of the selection strategy is to identify a node in the game tree that should be investigated further

- selection will select a node that has not yet been fully expanded
- **exploration vs. exploitation** dilemma: only exploring the best options identified so far further may lead to overlooking better options
- using information gained from random play, a statistical method for selection can be applied
  - popular choice: **Upper Confidence Trees (UCT)** (Kocsis and Szepesvári 2007) \(\leadsto\) select node with highest UCT value
• idea: adapt **Upper Confidence Bound (UCB)** for **k-armed bandit problems** to tree search

• k-armed bandit problem: Given a set of $k$ slot machines with unknown distributions of revenues, determine a strategy that will maximise revenue

• UCT approach: keep track of average revenues achieved and select the arm that yields the best upper confidence bound

$$\arg \max_i \tilde{V}_i + \sqrt{\frac{2 \ln N_i}{n_i}}$$

$\tilde{V}_i$: average value of node $n_i$

$N_i$: total number $n_i$’s parent has been visited during selection

$n_i$: total number $n_i$ has been visited

For deduction of formula and analysis, see Kocsis and Szepesvári (2007)
• various variation of the given formula have been considered to achieve different variations of the exploration-exploitation balance

• often an empirical evaluation is performed to identify the best-performing option

• two popular variations:
  • \( \text{arg max}_i \tilde{V}_i + c \cdot \sqrt{\frac{\ln N_i}{n_i}} \) with \( c \sim \sqrt{2} \) as tuning parameter
  • instead of \( \text{arg max} \) (“greedy”), only apply \( \text{arg max} \) with probability \( (1 - \epsilon) \) and otherwise draw \( i \) from uniform distribution (“epsilon-greedy”)

• \( \Rightarrow \) MCTS is still an active area of research!
Objective of expansion is to increase the considered part of the game tree.

- consider node selected in selection step
- selected node is not yet fully expanded
- partially expand the selected node, e.g., by randomly choosing one (two) moves to create [a] new child node(s)
Given a node in the game tree, the objective of rollout is to simulate a game **until a decisive state is reached**.

- possible approach: randomly select a move until game is decided (leaf node is reached), called **random rollout policy**
- for very deep game trees, a cutoff depth may be set instead of expansion until leaf nodes
- more sophisticated methods have been developed, yet random rollout still widely used
Once rollout has been performed for some node in the game tree, that node can be evaluated statistically. The objective of backpropagation is to **update evaluation of all parent nodes**.

- in case of UCT as simulation strategy: update UCT values $N_i, n_i, \bar{V}_i$ at all nodes on the path up to the root.
• Games offer a wide range of important microcosms for conducting AI research
  • game applications relevant too
• adversarial search is an important technique for two-player games such as chess or Go
• game trees represent state spaces in games
  • typically too large to search exhaustively
• MinMax algorithm allows determining best moves if exhaustive search possible
• $\alpha - \beta$ pruning and state evaluation heuristics allow us to search for ‘good’ moves in game trees
• state-of-the-art gameplay is based on Monte-Carlo Tree Search
